ZED-42 Adventure Game

Preliminary Specification

Patrick Broman, *d00-pbr@d.kth.se*
Tommie Gannert, *d00-tga@d.kth.se*
Josef Grahn, *d00-jgr@d.kth.se*
Mikael Hedberg, *d00-mhe@d.kth.se*
Johan Ljunglid, *d00-jlj@d.kth.se*
Niklas Lundström, *d00-nlu@d.kth.se*
Thomas Westerbäck, *u19e52s2@d.kth.se*

Project Initiator: KTH-GA's Yashar Moradbakhti

February 2003

Abstract

This document is an introduction to the adventure game ZED-42. It is a game constructed for the Game Awards competition in Spring 2003.

The document has three main parts. The first part is the general introduction to the game and it's background. It also documents the system requirements for the final game. The second part contains the bulk description, including a draft manuscript of the adventure, the modules of the game design and a sketch of the user interface.

In the last part, organization and administrative duties are discussed. This includes team organization, design paradigm, documentation and risk analysis. (For ease of reading, the risk analysis is a separate chapter.)

Note that this document is a draft of the game design. It is not intended as a user manual.

Contents

1 Introduction **3**
- 1.1 Background and Purpose . . . 3
- 1.2 Requirements and Limitations . . . 3
  - 1.2.1 Functions . . . 4
  - 1.2.2 Hardware and Software . . . 4

2 Solution Design **5**
- 2.1 System Diagram . . . 5
- 2.2 Modules . . . 5
  - 2.2.1 Game Logic . . . 5
  - 2.2.2 Script Engine . . . 5
  - 2.2.3 Animation . . . 5
  - 2.2.4 User Interface . . . 5
  - 2.2.5 Graphics Rendering . . . 6
  - 2.2.6 Physics Simulation . . . 6
  - 2.2.7 Resource Management . . . 7
  - 2.2.8 Error Handling . . . 7
  - 2.2.9 Sound Playing . . . 7
  - 2.2.10 User Input . . . 7
  - 2.2.11 Mathematics . . . 7
- 2.3 Story Outline . . . 7
- 2.4 User Interface . . . 8

3 Organization **10**
- 3.1 Method . . . 10
- 3.2 Administration . . . 10
- 3.3 Responsibility Distribution . . . 10
- 3.4 Project Schedule . . . 11
- 3.5 Documentation . . . 11

4 Risk Analysis **13**

Chapter 1

Introduction

Our aim is to construct a game which combines classic adventure game elements with the possibilities offered by modern PC hardware. We strive to provide the player with a more flexible and creative environment than has traditionally been offered in this genre, avoiding unmotivated limitations and allowing problems to be solved in more than one way.

While adventure games tend to use 2D graphics, ZED-42 will be in full 3D. However, we strive to maintain the overall feel of the classic point 'n' click adventure game.

Realizing our limitations, most of which are imposed by a tight schedule beyond our control, we have no ambition to make a game that is totally ground breaking and revolutionary in any field, but we are planning to introduce several improvements to the current adventure game standards, in areas ranging from graphics to story telling.

1.1 Background and Purpose

KTH-GA – KTH Game Awards – is a competition where the contestants create game demos for the general public. The games are then judged from their appearance, their feeling, and their documentation. What we seek to create is a game, not too hard to implement, but that could still fit in this course.

During our first meetings, several ideas were discussed. After settling with the adventure game idea, the subject was changed to what the story would be about. Among the suggested stories there were mostly Greeks, ghosts and robots. We decided to implement the robot idea, mostly because it was the easiest choice to design.

1.2 Requirements and Limitations

The goal of the project is to be able to present it in the KTH-GA sessions in the end of April. For this it is only required that the game is a demonstration copy of the game. This imposes two important restrictions: The game does not have to be complete, and it does not require to run on every platform available. This section deals with the requirements and limitations of ZED-42.

1.2.1 Functions

In the game you are ZED-42, a quite anonymous little factory worker robot that finds himself experiencing more excitement than he was programmed for.. as far as he knows, anyway. He embarks on an epic adventure in a futuristic world populated by robots, unraveling secrets about himself and the world around him, exploring a wide variety of environments and meeting many eccentric, dangerous and funny characters.

When first entering the scene, you as the player will be given the chance to explore an entire new world with as few limitations to the degree of freedom (also known as DOF) as possible. This is one of the reasons to use a 3D-accelerated game engine. The player will be able to pick things up, to use them and to move in the surrounding environments.

1.2.2 Hardware and Software

The game will be demonstrated in public during the KTH-GA demo day. For this the competitors may choose any hardware to play on. Because of the high 3D complexity of the game, the Microsoft Windows platform will be the target platform, although very little in the game is platform dependent. A 3D accelerator will most certainly be necessary for playing the game without getting frustrated.

Chapter 2

Solution Design

2.1 System Diagram

The ZED-42 game system is built up mainly of two parts, the engine and the driver part. The engine is a framework with the project name Raven. Around the Raven framework is all the driver code with logic, user interface, startup, and anything else specific to the game itself. A diagram of these two parts and their contents in form of modules can be found in figure 2.1.

2.2 Modules

2.2.1 Game Logic

The game logic consists mainly of interactable objects, which are objects in the world of the game, and events, which are things that happens in the game, possibly affecting objects in the game. Events are triggered by actions of the player and by other events. Every event also has a set of conditions which must be fulfilled before the event takes place. An action might have different results depending on what has happened earlier.

2.2.2 Script Engine

All information about the Events, Objects and Triggers will be specified in a script file that is read on startup of the game.

2.2.3 Animation

The animation module is responsible for animating the scene as a reaction to events that happen in the game logic.

2.2.4 User Interface

The implementation should separate graphical interface from functionality. There should be a set of actions that can be executed regardless of the interface (which only exists to make these actions available). The actions should be designed to fit within a clearly defined interface, i.e. that the set of actions

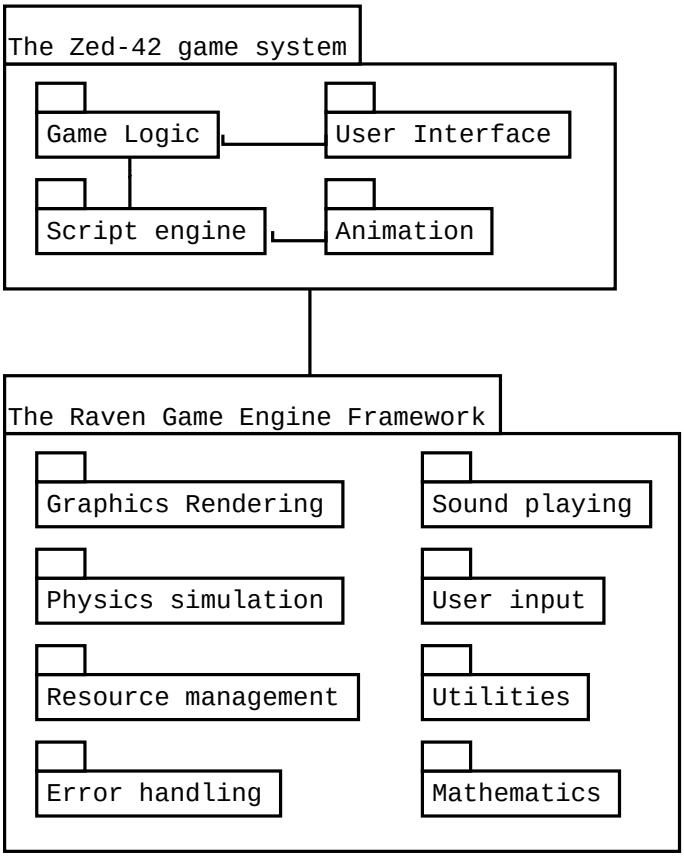

Figure 2.1: Diagram of subsystems in ZED-42

should not be larger than one can reasonably expect it to be possible to present with an arbitrary user interface. If a possible interface can not handle all the actions simultaneously, interface functions could be expanded and turned more abstract in order to accommodate all features.

2.2.5 Graphics Rendering

Rendering the graphics is the responsibility of the rendering module. It consists of classes that take a description of a 3D scene and turn it into a picture on the screen.

2.2.6 Physics Simulation

The physics simulation system will be responsible for checking if the line created into the scene when a user clicks somewhere intersects with some object in the scene, i.e. find out what the user has clicked on.

2.2.7 Resource Management

The resource management module takes care of the loading of resources from disc and manages already loaded resources.

2.2.8 Error Handling

The error handling module is responsible for making a list of all the unhandled errors and relaying them to the right module for handling.

2.2.9 Sound Playing

The sound subsystem is based around emitters and the receiver. The receiver receives the sound sent by the emitters, which is modulated depending on where the sound source (the emitter) is and how it's moving relative to the receiver.

The sound system is then responsible for taking these sound emitters and use their sound buffers to render sound onto a hardware sound output device.

2.2.10 User Input

The user input module is responsible for taking keyboard and mouse input and relaying it to the user interface module for handling. This module is also known as "controller".

2.2.11 Mathematics

The mathematics module contains all necessary mathematics classes needed for doing e.g. vector math, matrix math or quaternion calculations.

2.3 Story Outline

ZED-42 is a little worker robot. For as long as he can remember, he's been standing at the same assembly line, drilling identical holes in identical parts. He doesn't really mind, though. He's not programmed to. Even if he was, there wouldn't be very much he could do about it, since his movement is restricted by a power cable connecting him to the local power supply. Ironically, the very same cable will soon be ZED-42's ticket to freedom.

The story starts off with cut scene. A bunch of robots are standing in some sort of conference room, addressed by a human from a television screen. They are informed that the *xxx* robots in section *yyy* are to be terminated. Some of the robots look at each other, but soon they are nodding in unison with the other ones.

One day, as our little robot is busy drilling holes, he notices his left optical sensor is slightly misaligned. He complains a little about it under his breath, overheard by his neighbor robot — a senior colleague that's been around the block and always has more or less interesting advice to offer.

"Kid, you better get that fixed. You've gotta stand up for yourself, y'know? Don't let them push you around, boy — if your sensor's misaligned it's the maintenance robot's Man Damned responsibility to fix it. See that lazy pile o'

junk there in the corner? That's the maintenance bot. Get him over here and he should have you back in tip-top shape in no-time."

The maintenance robot is standing just across the room, so ZED-42 decides to call him. The maintenance robot looks confused, as if he had been lost in thought, but once he's figured out who was calling him and why, he starts moving towards ZED-42. Unfortunately, the maintenance bot just had his new wheels installed, so his precision leaves much to be desired. Instead of stopping next to ZED-42, he trips over his power cable, and leaves ZED-42 entangled in it. ZED-42 repeatedly tries to untie himself, but manages to end up more tied up each time. He screams at the maintenance robot, ordering him to free him immediately. The maintenance robot attempts are futile, however, and seeing ZED-42 getting ever angrier forces him to offer another solution (which he isn't really allowed to do) — namely, to give ZED-42 a battery back and disconnect him temporarily. He does, and just as the cable is disconnected, two robots enter the room.

We recognize the newly arrived robots from the cut scene, and they announce their assignment — to disconnect all the robots in the room. They will give the robots a few minutes alone to say their goodbyes, and when they return, recycling will commence. As they leave the room, ZED-42 is given an opportunity to hide. Since he is no longer connected, he can move about freely. Once he has managed to hide behind the door, the "executioner" robots will return, and start disconnecting ZED-42's friends. ZED-42 now has the chance to escape through the open door. Once he does, there is a small cut-scene. Just as he steps outside the door, a cleaning robot comes and kicks ZED-42 down the disposal tube. The other end turns out to be located outside the factory, and leads to a container. So, our hero lands in the container. Now he has to get out.

2.4 User Interface

Executing an action, e.g. to examine an object, is done by right clicking with the mouse. Then a menu with different icons will appear. Then the player will push the icon that executes the desired action.

There will be as few icons as possible and they will be comprehensive. Then pushing a certain icon may cause different actions at different situations. For example, pushing the "hand-icon" may cause that the robot picks up an object, but could also cause the robot to cuddle the "robot-cat". To avoid ambiguity if there are many actions that could be executed with one icon in one situation, the icon will possibly show sub menus at that situation.

Below is shown some icons and actions that could be executed:

- "eye-icon": look at
- "hand-icon": pick up, move and use tools, cuddle and touch objects
- "mouth-icon": communicate

Other actions that should be available are: start/stop, drag, open/close, give.

To be able to, for example, save, load, pause, or exit the game and to settle the settings concerning sound, brightness and graphic the player have to push a function-button e.g. F1.

To be able to reach ”inventory” the player will have to drag the mouse-arrow at the very bottom of the screen. Then all the inventories will appear. Now the player can reach and possibly combine different inventories. I works sort of like the icon row of MacOS X.

Figure 2.2: A design sketch for the user interface

Chapter 3

Organization

Since the project includes a large number of very different tasks, we have decided to categorize the work into separate areas and to assign a supervisor for each area. The supervisor is responsible for assembling a work group and to make sure all necessary work is done in his area.

To ensure that crucial design decisions can be made even when opinions differ among the project members we have also appointed a group member as technical lead and one as art lead with the authority to force a decision if necessary.

Of course, we also have a project supervisor that coordinates the whole project and makes sure every task is assigned a supervisor.

3.1 Method

We have chosen to work mainly according to the eXtreme Programming scheme, although with two exceptions. We will usually not write program code in pairs, and some parts of the code (i.e. the game engine) is not regarded as the property of every project member.

3.2 Administration

Coordination between the group members is, apart from informally during our work, managed with weekly meetings and by using a Wiki page (i.e. a web page that any member can edit in an easy way).

3.3 Responsibility Distribution

The project's many parts have led to the need for supervisors of some of these. They are all listed in Table 3.3.

Project Supervisor: Josef Grahn

Technical Lead: Mikael Hedberg

Art Lead: Patrick Broman

Engine Design: Josef Grahn, Mikael Hedberg

Game Logics: Niklas Lundström

Sound: Mikael Hedberg

Music: Thomas Westerbäck

Documentation: Tommie Gannert

Story: Niklas Lundström

Textures: Johan Ljunglid

User Interface: Patrick Broman

Table 3.1: Team members responsible for each part

3.4 Project Schedule

An overview of the project schedule can be seen in Fig. 3.4. This is a rough planning of the different phases and modules of the project.

3.5 Documentation

All work done in the project will be extensively documented. All code should be well documented using JavaDoc-style comments allowing us to auto generate documentation using Doxygen.

All work not concerning code should also be well documented with, at least, a document saying what has been done and how it was done, and if necessary why it was done.

Documents should be written in parallel with development. If something is changed in our work the corresponding document should also be changed, possibly with a note of what was before the change.

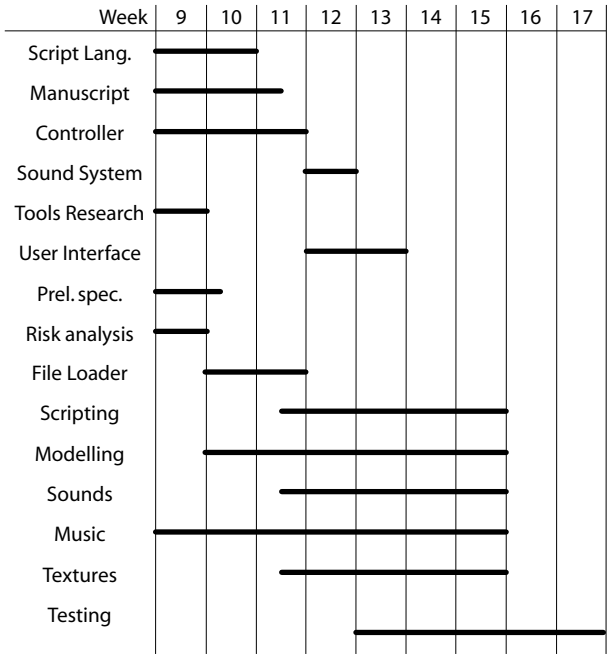

Figure 3.1: Project Schedule, Weeks Overview

Chapter 4

Risk Analysis

There are rather few risks that we deal with in this project. This is due to the fact that we have only two requirements for the production: To deliver in time, and to make it show *something*. In the following list, we have tried to write down the worst risks, in decreasing probability order.

1. **Wrong priority assignment.** With this we mean that too much time is spent on one thing (perhaps unnecessarily) while others, more important parts, are discarded. The solution, we hope, is based on the personal responsibility, where each person on the project has a responsibility for one small part of the project. The drawback of this might be the tendency of programmers to concentrate on parts that are fun to implement.

2. **Bad scheduling.** It will not matter how much we try to be on the time schedule of the project. We will fail. The best we can do is to fix the critical path through the project and hope that at least that path will be correct. To minimize the risk of scheduling gaps, regular meetings will be held to update each project member on the status.

3. **Too many ideas.** The game we are to implement is a game *demonstration* of a big game. It doesn't need all features or odd bolts. This also means that we must limit our implementation and design to specific parts that are to work well during the demonstration. Again, there is a risk that too much time is spent on low priority tasks.

4. **No collaboration.** Without regular meetings there is a risk of tasks being done by individual team members without others knowing they are being taken care of. This ultimately leads to task duplication and drift in schedule.

5. **Forgotten details.** A consequence of prioritizing in a bad way, things may be forgotten. Thus, as above, scheduling is everything, since knowing *when* to do something requires that we know *that* it should be done.

6. **Ill team member** The game is build around a 3D game engine called Raven. This is a closed-source sub project of ZED-42 with only two members working on it. If one of them would get ill or sick, the whole project may be in jeopardy.